VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Stereo korespondence

Stereomatch

2011 **Jakub Stonawski**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jakub Stonawski**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Stereo korespondence
Stereomatch

Zásady pro vypracování:

Stereo korespondence je jedním ze základních problémů v oblasti rekonstrukce 3D scény. Jedná se o těžký problém, který doposud není uspokojivě vyřešen. Nalezení korespondence se často formuluje jako globální optimalizační problém, který zahrnuje všechna možná geometrická a fotometrická omezení jako je například zákryt nebo průhled.

1. Seznamte se s algoritmy stereo korespondence (SAD, SSD, Census, Rank).
2. Implementujte vybraný algoritmus pro stereo korespondenci.
3. Proveďte optimalizaci pro GPU procesory využívající architekturu CUDA.
4. Navrhněte metodu porovnávající jejich úspěšnost pro zadané dvojice snímků.

Seznam doporučené odborné literatury:

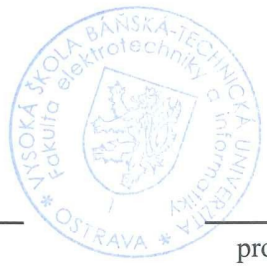
Richard Hartley, Andrew Zisserman, Multiple View Geometry in Computer Vision, 2004, ISBN 978-0521540513
Boguslaw Cyganek, An Introduction to 3D Computer Vision Techniques and Algorithms, 2009, ISBN 978-0470017043
Rafael C. Gonzalez, Digital Image Processing, Prentice Hall; 3 edition, 2007, ISBN 978-0131687288
http://vision.middlebury.edu/stereo/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Michal Krumnikl**

Datum zadání: 19.11.2010

Datum odevzdání: 06.05.2011



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2011 Jakub Stonawski

Na tomto místě bych především rád poděkoval svému vedoucímu bakalářské práce panu Ing. Michalu Krumniklovi za jeho cenné rady a čas při tvorbě této práce a také všem blízkým za podporu.

Abstrakt

Tato bakalářská práce si za hlavní cíl klade osvětlení základních metod, které se využívají při stereo korespondenci, nástinů možných řešení problémů, které se u těchto metod vyskytují a také optimalizaci algoritmů s využitím technologie CUDA.
Základním cílem stereo korespondence je analyzovat 2 obrazy, zaznamenávající shodnou scénu, ovšem s odlišnými úhly pohledu a nalézt v nich stejné prvky. Tohoto lze poté dále využít při rekonstrukci 3D scény. Výpočetní nároky na takovéto analýzy jsou ovšem značně veliké, a tak je třeba hledat možnosti optimalizace, kterou přináší například právě CUDA technologie.

Klíčová slova

Stereo korespondence, CUDA, Disparitní mapa , Optimalizace pomocí GPU

Abstract

Main goal of this bachelor work is explain main methods that are used for stereomatch, some basic solutions of problems that can appear during using of these methods and also propose optimization of these algorithm with help of CUDA technology.
Stereomatch's basic purpose is to analyze 2 images that shows same scene but from different angles of view and find common elements. By means of this principle we can then reconstruct 3D scene. But basic problem is that the computation requirements required for this can be abnormal. Thus we can find and use some optimization, that in our case can be application of CUDA technology.

Keywords

Stereomatch, CUDA, Disparity map, Optimization with GPU

Seznam použitých zkratek a symbolů

BP	Bayesian belief Propagation
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
GFLOPS	Giga Floating-point Operations per Second
GPGPU	General-Purpose computing on Graphics Processing Units
GPU	Graphic Processing Unit
MIT	Massachusetts Institute of Technology
MRF	Markov Random Fields
SSD	Sum of Squared Differences
SAD	Sum of absolute Differences
ZSSD	Zero mean Sum of Squared Differences

Obsah

1 Úvod

Už od přírody je člověk obdařen schopností vnímat celý okolní svět v prostorovém, tedy troji-dimenzionálním, obrazu. I když je pro většinu z nás tato vlastnost brána za zcela automatickou, fakticky se každý člověk této schopnosti musí v začátku svého života naučit. Respektive musí se schopnosti vytvořit a zpracovat 3D obraz naučit náš mozek. Kdyby se z jakéhokoliv důvodu stalo, že by se mozek člověka této schopnosti nebyl schopen naučit, nebyl by tento člověk prakticky vůbec schopen prostorového vnímání. Jeho oči by sice mohli obraz zpracovávat bezchybně, ale mozku by předávali pouze 2 obrazy okolí z různých úhlů pohledu ale mozek by je do výsledného prostorového obrazu nebyl schopen zpracovat.

K tomu aby jakýkoliv tvor, ať už přímo člověk nebo jiný živočich byl schopen vnímat okolní svět prostorově je zapotřebí 2 základních prostředků – orgánu, který mu umožní zachytit obraz okolního světa a mozku, který tento „vstupní signál“ dokáže proměnit do podoby prostorového obrazu.

Co se týká prvního prostředku, tedy v našem případě lidských očí, není jen potřeba triviálně zachytit obraz okolního světa. K tomu, aby z takovéhoto zachyceného obrazu bylo možno vytvořit 3D obraz je zapotřebí, aby byly oči umístěné ve správné vzdálenosti od sebe. Tento oční rozestup se u lidí běžně pohybuje v rozmezí od 56 do 72 mm. Tato hodnota zajistí , pro nás ideální, rekonstrukci prostorového obrazu. Kdybychom tuto hodnotu měli výrazně vyšší, jako je tomu například u některých druhů zvířat, naše prostorové vnímání by se znatelně zhoršilo, ovšem na druhou stranu bychom byly schopni zachytit větší celek obrazu okolí. Obdobně kdybychom tuto hodnotu očního rozestupu měli nižší, naše vnímání prostoru by se sice ještě více zlepšilo, ovšem na úkor velikosti celku vnímaného obrazu.

Samozřejmě v oblasti počítačové vědy je již dlouhou řadu let snaha převést tento princip do elektronické podoby a umožnit tak i počítačům možnost „naučit se“schopnosti vidět prostorově a orientovat se v takovémto prostoru.

Není dnes problém poskytnout vynikající snímání obrazu při použití kvalitních kamer či čoček fotoaparátů a tyto poté rozmístit přesně tak, aby simulovaly výhled z našich očí. Problém, se kterým se dnes potýkáme je v možnosti poskytnout dostatečný výpočetní výkon aplikacím tak, aby byly schopny v co možná nejkratším čase zpracovat obrazy a vytvořit z nich 3D rekonstrukci scény.

Tomuto problému bych se tedy chtěl věnovat v této bakalářské práci. V dnešní době existuje celá řada algoritmů, které popisují způsob, jak lze jednotlivé obrazy porovnávat a vytvářet z nich disparitní mapy, které nakonec lze využít při rekonstrukci 3D scény.

Snahou této práce je popsat základní principy stereo korespondence, uvést základní algoritmy, které si při ní dají použít a také optimalizovat vybrané algoritmy pomocí technologie CUDA, která poskytuje výpočetní výkon grafického procesoru k potřebným výpočtům a manipulacím s obrazovými daty.

V závěru této práce jsou uvedeny výsledné úspěšnosti implementovaných algoritmů, podle kterých lze usoudit, který algoritmus je pro optimalizaci při použití CUDA architektury nejvhodnější.

2 Rekonstrukce 3D scény

Problém rekonstrukce 3D scény není nijak nový. Prvními lidmi, kteří se zabývali problematikou prostorového vidění byli již Antičtí Řekové a pravděpodobně první prací, která byla na téma disparity sepsána, byl spis od Aristotela z roku 380 př.n.l.[1]. I v dnešní počítačové době je stále tento problém aktuální a zejména rozvoj robotiky a vůbec celková snaha umožnit počítačům analyzovat svět způsobem, jakým to dokáže člověk posouvá nároky v této oblasti do stále větších rozměrů.

Jedním z prvních vědců, kteří se začali v novodobé historii počítačových věd tímto problémem zabývat byl kupříkladu David Marr, který jakožto pracovník laboratoře umělé inteligence na MIT v 80. letech navrhl na téma počítačového vidění svou teorii, kterou i později využíval při řešení stereo korespondence. Právě v této teorii upozorňoval na fakt, že zpracování obrazů pomocí lidského vizuálního systému má velice komplikovanou hierarchickou strukturu, která vyžaduje vícevrstvé zpracovávání obrazu a že převedení tohoto systému do počítačového zpracování nebude vůbec jednoduché, což se ostatně prokazuje dodnes, kdy se stále vyskytují problémy vyplývající z geometrických či fotometrických omezení[2].

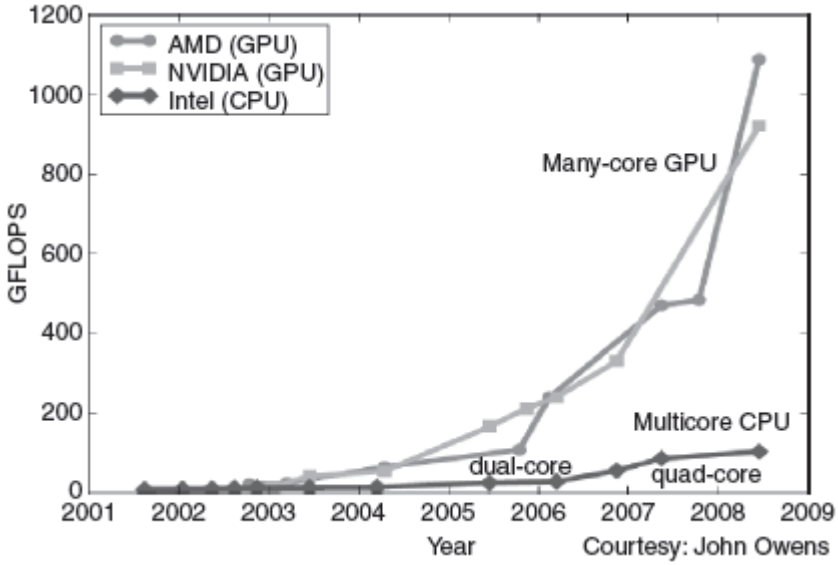
Dnes pro nás již není problém rekonstruovat prostorovou scénu z 2 či více obrazů, které danou scénu zachycují s různých úhlů, samozřejmě za předpokladů kvalitních vstupních dat (např. minima objektů, které jsou v zákrytu a pod.) a dosáhnout poměrně uspokojivých výsledků. Dnešním, do této chvíle ještě ne zcela uspokojivě vyřešeným problémem je rekonstruovat takovouto scénu co možná nejrychleji a nejefektivněji tak, abychom mohli poskytnout prostorové vidění programům v pokud možno reálném čase. Tyto potřeby ovšem přinášejí velmi vysoké nároky na výpočetní výkon, který je k takové rekonstrukci trojrozměrných scén zapotřebí.

2.1 Možnosti realizace výpočetních operací

V počátečních letech vývoje aplikací zabývajících se problémem rekonstrukce 3D scén bylo zvykem využívat k výpočetním operacím zejména procesory CPU, protože se jednalo o více méně jediný prostředek, který byl schopen složité výpočetní operace realizovat. Navíc výkonost CPU se v průběhu posledních více než 20 let stále prudce zvyšovala a v současné době se u běžných desktopových počítačů pohybuje v řádech jednotek FLOPS (u serverových počítačů dokonce v řádech stovek FLOPS)[3].

Ovšem v současné době je vhodnější používat na tyto náročné výpočetní operace, které jsou dělány nad obrazovými daty grafický procesor GPU (tento přístup využívání GPU se v literatuře běžně označuje zkratkou GPGPU). Nynější grafické karty totiž již výkonností svého grafického procesoru často mnohonásobně předčí výkony dnešních CPU.

Dnešní CPU, se totiž při zpracování instrukcí orientují na tzv. multicore přístup, tzn. zachovávají stávající běžné rychlosti vykonávání sekvenčních příkazů programů, ale jednotlivé úkony rozdělují do více jader. Naproti tomu grafické procesory při zpracování instrukcí využívají přístupu many-core. Tento druh přístupu se více orientuje na možnost poskytnutí lepší paralelní propustnosti jednotlivým aplikacím.



Obr. 2.1.1: Vývoj výkonnosti CPU a GPU (podle hodnoty FLOPS)

Zatímco u první metody přístupu multicore u CPU se dnes běžně setkáváme s procesory s počtem 2 až 8 jader, u metody many-core využívané grafickými procesory se počty jader pohybují v řádech stovek. Jádra GPU procesorů jsou sice jednodušší a menší, tím pádem oproti procesorům CPU jsou omezenější v poskytování vnitřní logiky, ale poskytují daleko lepší hodnoty FLOPS (viz Obr. 2.1.1)[3] a pro naše výpočetní operace, které budeme při stereo korespondenci využívat je tedy výrazně vhodnější optimalizovat programy pro grafické procesory.

2.2 Stereo korespondence a určení disparity

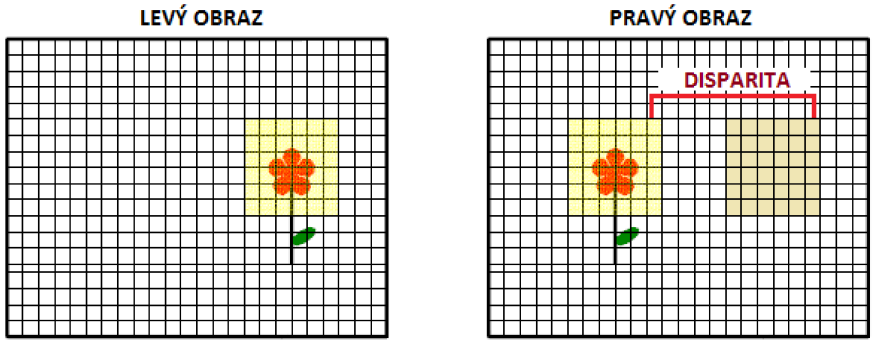
Stereo korespondence je hlavním nástrojem při rekonstrukci 3D scény. Jejím hlavním úkolem je analyzovat 2 nebo dokonce více obrazů, které zachycují totožnou scénu z mírně odlišných úhlů pohledu a najít v těchto obrazech společné elementy. Právě nalezením těchto totožných elementů jsme totiž schopni pomocí rozdílných algoritmů vypočítat disparitu a vytvořit tzv. disparitní mapu, ze které jsme schopni určit prostorové podrobnosti celé scény, jako je například vzdálenost jednotlivých objektů od sebe nebo od pozorovacího místa objektivů a tímto i fakticky celou tuto scénu prostorově vymodelovat.

2.2.1 Princip stanovení disparity a výpočet vzdálenosti objektů

Jak je výše uvedeno, tím že scéna je zaznamenána pomocí 2 obrazů z mírně odlišných úhlů (respektive pro zpřesnění celkového výsledku je vhodné používat více obrazů, z jedné kamery či fotoaparátu, ale to pro vysvětlení principu nemusíme uvažovat), stejný objekt, který je zaznamenán na prvním obraze nebude umístěn na stejném místě i na druhém obraze. A právě tento rozdíl umístění stejného objektu na 1. a 2. obraze udávaného v pixelech je disparita (viz Obr. 2.2.2)[4].

Tato hodnota disparity je pro pozdější tvorbu disparitní mapy nebo dokonce rekonstrukci celé prostorové scény stěžejní věc, a proto se na přesné určení této hodnoty musíme při tvorbě algoritmů stereo korespondence zaměřit.

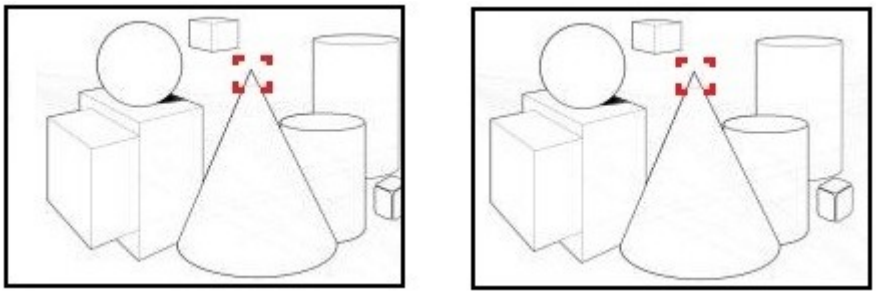
Obecný postup, který můžeme pro určení disparity navrhnout bych tedy dále vysvětlil. Pro jednoduchost budu při tomto příkladu uvažovat za vstupní data pouze 2 obrazy scény (levý a pravý). První důležitou věcí je určit, který ze 2 vstupních obrazů označíme jako referenční. Pomocí tohoto referenčního obrazu budeme totiž vzápětí analyzovat druhý obraz a hledat v něm stereo korespondenci.



Obr. 2.2.2: Určení disparity

V počáteční fázi vybereme určitou oblast v referenčním obraze, kterou následně budeme za pomocí určitého algoritmu stereo korespondence hledat v obrazu druhém. Důvod, proč vybíráme oblast více pixelů a neporovnáváme jednotlivé pixely navzájem je jednoduchý. Snažíme se najít určité stejné objekty v obrazech a pouhým porovnáním jednotlivých pixelů navzájem bychom úspěšného výsledku nebyly schopni dosáhnout. Viz Obr. 2.2.3.

Pokud bychom porovnávali jednotlivé pixely navzájem, určitě bychom neuspěli, protože jen např. stejných pixelů bílé barvy bychom nalezli mnoho. Pokud ovšem budeme hledat pixel se svým okolím (na obrázku zvýrazněno červeným rámečkem) naše šance se bezesporu zvýší. Nesmíme ovšem velikost oblasti zvolit ani příliš velikou, protože toto by nám výpočet znatelně nezefektivnilo.

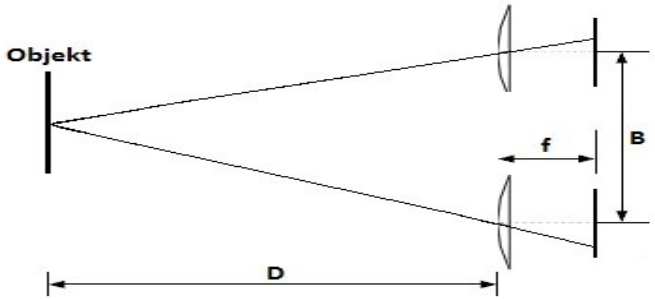


Obr. 2.2.3: Nalezení shody v levém a pravém obraze

Sekvenčně budeme procházet všechny pixely (a jejich okolí dle velikosti vybrané oblasti referenčního obrazu) v druhém obrazu a hledat shodu.

Tento způsob analýzy obrazů se nazývá korelační zpracování obrazu a jelikož ho v programu, který je k této bakalářské práci přiložen využívám, budu se zabývat pouze tímto způsobem. Pro úplnost doplním, že druhým hlavním způsobem analýzy obrazů je tzv. příznakové zpracování obrazu, které se využívá zejména v robotice a neanalyzujeme u něj všechny pixely obrazu, ale hledáme určité snadno detekované prvky (např. hrany), které poté porovnáváme.

V momentě, kdy nalezneme objekt, který je podle našeho algoritmu nejpravděpodobněji shodný s objektem v referenčním obraze lze již tedy snadno určit výslednou disparitu.



Obr. 2.2.4: Princip stanovení vzdálenosti objektu od kamer

Pomocí této určené disparity **d** a dalších základních parametrů, jakými jsou ohnisková vzdálenost kamery či fotoaparátu **f** a vzdálenost kamer či fotoaparátů od sebe navzájem neboli účaří **B** (viz Obr. 2.2.4) jsme nakonec schopni vypočítat vzdálenost objektu od kamer pomocí následujícího vzorce[4]:

$$D = \frac{Bf}{d}$$

Vzorec 1

2.2.2 Tvorba disparitní mapy a její druhy

Princip nalezení stereo korespondence a určení disparity je již tedy objasněn. Našim cílem ale není pouhé nalezení disparity u jednotlivého pixelu, konečným výstupem, kteréhokoliv algoritmu stereo korespondence by měla být disparitní mapa, z jejíž pomocí jsme schopni vnímat hloubku dané scény a popř. scénu zrekonstruovat (viz Obr. 2.2.5).

Disparitní mapu si můžeme představit jako matici **M** o rozměrech stejných, jako původní analyzované obrazy. Tedy pokud mají vstupní obrazy(L,R) rozměry **w,h** (samozřejmě musí platit že $\mathbf{w_L}=\mathbf{w_R}$ a $\mathbf{h_L}=\mathbf{h_R}$) výsledná disparitní mapa (matice M) bude mít rozměry taktéž w,h. Hodnota prvku **p** v disparitní mapě na pozici **x,y** je potom rovna disparitě pixelu na stejné pozici **x,y** ve vstupním referenčním obrazu.

Obr. 2.2.5: Disparitní mapa (uprostřed) vytvořená ze 2 obrazů scény

Disparitní mapy můžeme rozdělit na 2 základní druhy. Prvním z nich je tzv. řídká disparitní mapa neboli sparse disparity map. Tento typ se většinou vytváří při příznakovém zpracování obrazu, o kterém jsem se již zmiňoval v odstavci 2.2.1. Výhodou této řídké disparitní mapy je zejména to, že ji lze vypočítat a sestavit za velice krátkou dobu, jelikož v obraze analyzujeme pouze určité snadno rozpoznatelné objekty (hrany, rohy) a ne všechny pixely. Hlavní nevýhodou je ovšem nízká kvalita takovéto výsledné disparitní mapy. Chybějící hodnoty disparity se totiž musí dopočítávat pomocí odhadu, které jsou založeny na disparitách pixelů, které s takovýmto chybějícím bodem sousedí.[5] Tento odhad hodnot vede ovšem často k nepřesným hodnotám disparit, které pak kvalitu výsledné disparitní mapy snižují.

Druhým typem mapy je tzv. hustá disparitní mapa neboli dense disparity map. Oproti řídké disparitní mapě je tato mapa výrazně kvalitnější. V husté disparitní mapě jsme schopni daleko lépe rozpoznávat hloubku obrazu a rozeznávat výsledné objekty. Bohužel tvorba takovéto mapy je výrazně pomalejší, jelikož ji tvoříme zejména při korelačním zpracování obrazu, při kterém porovnáváme všechny pixely v obraze.

Při tvorbě aplikace, která se bude zabývat stereo korespondencí nebo rekonstrukcí 3D scény bychom si tedy vždy měli dobře rozmyslet jak druh metody, pomocí které budeme obraz analyzovat tak i jaký typ disparitní mapy budeme vytvářet.

2.3 Možné využití techniky stereo korespondence

Možnosti využití techniky stereo korespondence v praxi jsou širší, než pouze rekonstrukce prostorové scény s 2 či více dvourozměrných obrazů, kterou lze např. využít v robotice při orientaci robotů v prostoru[6]. Vyhledávání společných či konkrétních prvků v obraze může být použito v celé řadě jiných aplikací, jako jsou programy pro detekci a rozeznávání lidských tváří, aplikací v automobilovém průmyslu, kdy můžeme docílit rozpoznávání nebezpečných objektů na silnici nebo dokonce dopravních značek či aplikací v lékařství, kdy lze detekovat určité tvary na rentgenových snímcích.

Pro všechny tyto praktické využití je ovšem nutné zajistit, nejen to, aby vykonávání programů bylo dostatečně svižné, ale zejména aby výsledky, které tyto programy budou poskytovat byly co nejpřesnější a nejkvalitnější. Zatímco stále lepší a lepší optimalizací stávajících algoritmů jsme schopni svižnějších běhů programů, bohužel do této doby nejsme schopni 100% zajistit detekci stereo korespondence u všech objektů v kterékoliv scéně. Stále se potýkáme s problémy, kdy jsou např. objekty scény v zákrytu jiných objektů či jinými geometrickými omezeními, takže práce v této oblasti je stále ještě mnoho.

3 Algoritmy využívané při stereo korespondenci

Algoritmů, které se dnes běžně využívají při určování stereo korespondence v obrazech je celá řada, můžeme je rozdělovat do rozdílných skupin podle toho, jakým způsobem pracují se vstupními obrazovými daty, jak kvalitní při jejich použití získáme výslednou disparitní mapu i podle jejich praktického využití v konkrétních situacích.

Společným prvkem všech těchto algoritmů je poté samozřejmě nalezení určité shody ve vstupních datech a její další zpracování, které vyvrcholuje například až v tvorbu disparitní mapy. Nalezení této shody lze chápat jako proces nalezení určité korelace, tedy nalezení určitého stupně vzájemného vztahu mezi vstupními daty ze dvou rozdílných skupin[5]. Právě tento proces je ovšem v oblasti počítačových věd stále neuspokojivě dořešen a stále se pracuje na vylepšování stávajících algoritmů či tvorbě nových, které mohou poskytovat zcela odlišný přístup k problematice.

V následujících částech této kapitoly bych se tedy zaměřil na možné způsoby rozdělení algoritmů, jejich výčet, hlavní rozdíly a situace, ve kterých je použití daných algoritmů vhodné a kde nikoliv.

3.1 Možné způsoby dělení algoritmů stereo korespondence

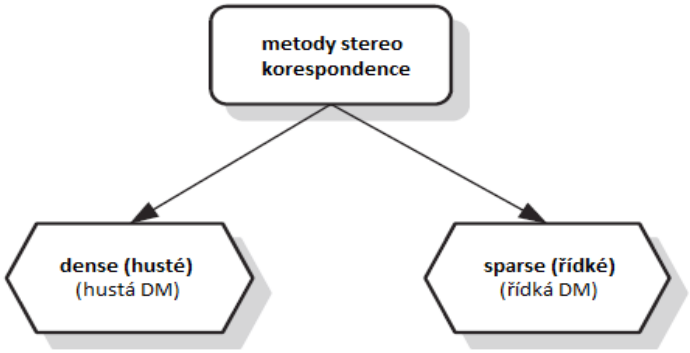
Jak již jsem uvedl výše, možných způsobů jak lze rozdělit současné algoritmy využívané při analýze stereo korespondence může být celá řada. Není jednoduché zavést určité standardizované rozdělení těchto algoritmů nicméně jsou určité hrubé základní kategorie, do kterých lze stávající algoritmy roztřídit a tím umožnit lepší orientaci v celé problematice. Tyto základní typy dělení metod využívaných při stereo korespondenci bych tedy dále popsal.

3.1.1 Dělení algoritmů v závislosti na výsledné disparitní mapě

Prvním způsobem, kterým jsme schopni všechny algoritmy používané při stereo korespondenci rozdělit je určením typu výsledné disparitní mapy. O dvou základních druzích disparitních map jsem se již zmínil v kapitole 2.2.2, takže zde jen pro připomenutí zopakuji, že existují disparitní mapy husté a řídké (dense, sparse), které se liší výslednou kvalitou zpracování vstupních obrazů. A podle těchto map probíhá tento první způsob dělení algoritmů (viz Obr. 3.1.6).

V literatuře se také můžeme setkat s pojmem dělení algoritmů na korelační a příznakové, což je dělení založené na stejném principu, jelikož při korelačním zpracování obrazu vzniká hustá disparitní mapa zatímco při příznakovém vznikne mapa řídká.

Našim cílem je samozřejmě to, aby celý výsledný program fungoval co nejpřesněji a poskytoval nejrelevantnější data. Takovýto výsledný efekt by nám poskytovaly právě ty algoritmy, které by vytvářely disparitní mapu hustou. Ta totiž poskytuje hodnoty disparity pro všechny (nebo pro většinu) pixelů vstupních obrazů. Kdybychom výsledný program chtěli například využívat pro rekonstrukci celé scény za pomocí syntézy jednotlivých obrazů, použití husté disparitní mapy by bylo nejvhodnější[7].



Obr. 3.1.6: Dělení algoritmů dle výsledné disparitní mapy

Na druhou stranu vytvoření takto kvalitní disparitní mapy vyžaduje, v porovnání s tvorbou řídké disparitní mapy, daleko větší výpočetní výkon a čas. Algoritmy stereo korespondence, které vytvářejí hustou disparitní mapu totiž musejí analyzovat každý pixel ve vstupních obrazech a vypočítávat pro něj hodnotu disparity. Oproti tomu při tvorbě řídké disparitní mapy algoritmy hledají určité snadno rozeznatelné objekty v obraze a disparitu určují pouze pro ně. Pro zbývající pixely vetšinou hodnoty disparity predikují.

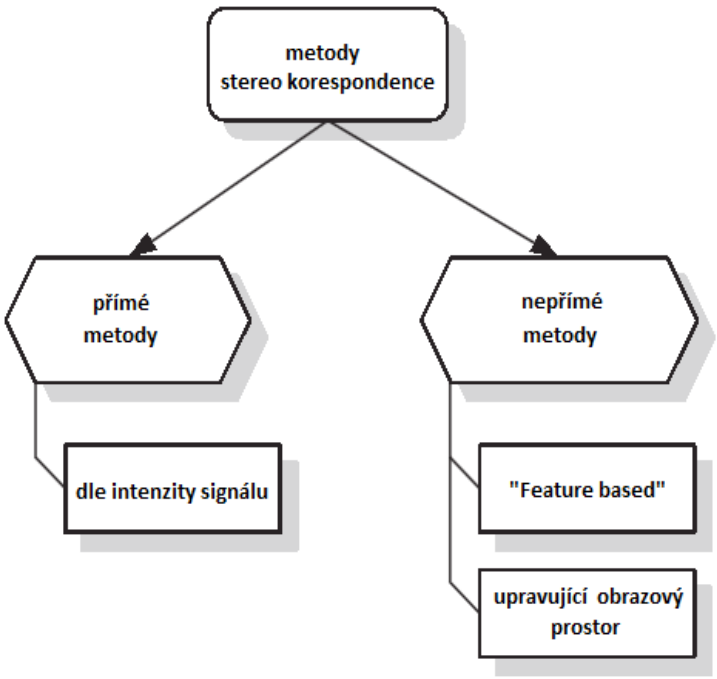
Vhodnost použití algoritmu vytvářejícího hustou či řídkou disparitní mapu je tedy poměrně jednoznačná. V případě, že náš program si klade za cíl především rychlost, což je hlavní požadavek kupříkladu u realtime zpracování videa je vhodné uvažovat o sparse metodách. Pokud by ovšem výstup z našeho programu měl sloužit pro co nejrealističtější popsání prostorové scény měli bychom určitě používat metody tvořící husté disparitní mapy.

3.1.2 Dělení algoritmů dle formátu vstupních dat

Dalším způsobem, kterým lze rozdělit algoritmy stereo korespondence je podle typu vstupních dat, se kterými náš algoritmus bude dále počítat. Možnosti jsou, stejně jako v předchozím způsobu dělení, dvě. V prvním případě algoritmy počítají výsledné hodnoty disparit v závislostí na hodnotě intenzity např. jasové složky každého pixelu. Jelikož přistupují k jednotlivým vstupním datům přímo ze vstupních obrazů aniž by je dále upravovali tyto algoritmy zařazujeme do kategorie přímých metod stereo korespondence.

V opačném případě, tedy tehdy, když se mezi fázi získání vstupních dat přímo z obrazu a fázi výsledného zpracování těchto hodnot vloží jakákoliv úprava vstupních hodnot, hovoříme o algoritmech z kategorie nepřímých metod stereo korespondence. Tuto kategorii nepřímých metod poté ještě můžeme rozdělit na podkategorie dle způsobu, jakým vstupní hodnoty upravují(viz Obr. 3.1.7).

Těmito úpravami mohou být kupříkladu transformace intenzity vstupních dat do jiných domén nebo výpočet jiných charakteristických vlastností vstupních dat podle kterých lze dále vyhledávat shodu. V kategorii nepřímých metod se poté mohou objevit například algoritmy, které analyzují počty permutací, které jsou zapotřebí k seřazení všech pixelů v určitém analyzovaném okně či algoritmy vypočítávající počty pixelů, které jsou ve zkoumaném okolí obrazu větší než vybraný pixel.



Obr. 3.1.7: Dělení algoritmů dle formátu vstupních dat

Co se týká vhodnosti použití algoritmů z kategorie přímých či nepřímých metod, není již toto posouzení zdaleka tak jednoduché, jako v předchozím případě. samozřejmě i zde bychom mohli určit, že v případě, pokud budeme analyzovat intenzitu pixelů přímo podle jejich jakových složek (a tedy využívat algoritmy z kategorie přímých metod) mělo by být zpracování znatelně rychlejší než v případě, že budeme před analýzou vstupní data jakkoliv upravovat.

Avšak znovu můžeme říci, že pokud budeme chtít získat co nejrelevantnější výsledky bylo by vhodné aspoň zvážit, zda-li není vhodné využít algoritmů ze skupiny nepřímých metod stereo korespondence. Znovu je tedy vhodné zvážit, z jaké skupiny zvolit algoritmus pro náš konkrétní problém.

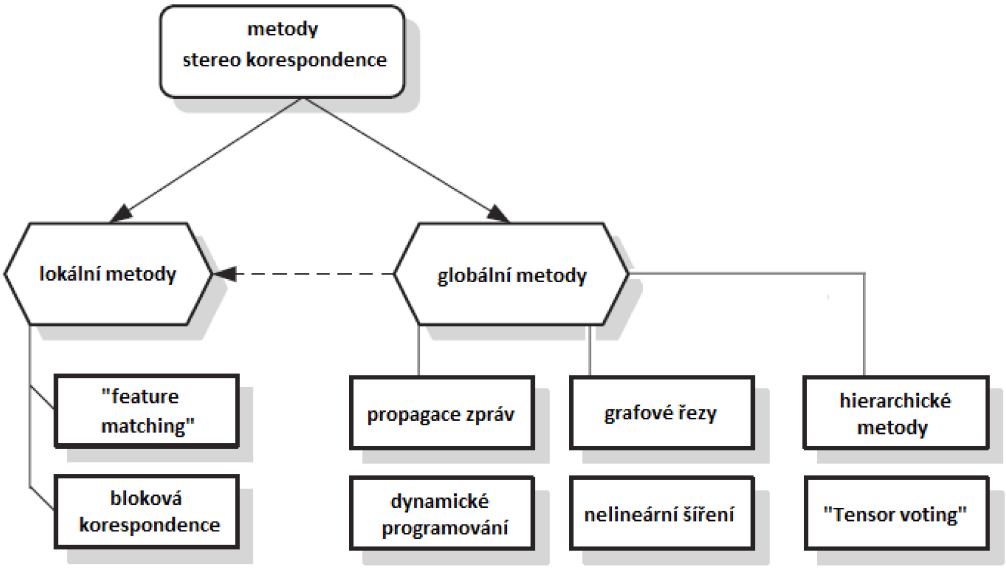
3.1.3 Dělení algoritmů dle velikosti dat potřebných pro určení disparity

Poslední základní metoda, podle které lze rozdělit algoritmy používané pro analýzu stereo korespondence je taková, že analyzujeme jak velkou oblast vstupních dat náš konkrétní algoritmus využívá k výpočtu disparity všech pixelů. Opět jako již tomu bylo u předchozích dvou dělení i toto vytváří 2 hlavní skupiny algoritmů.

První skupina analyzuje pouze určitou oblast pixelů, která se vyskytuje v okolí daného pixelů, pro kterého určujeme disparitu. Algoritmy využívající tohoto principu budou tedy zařazeny do první kategorie jako lokální metody stereo korespondence. V případě druhé skupiny algoritmů, tyto využívají pro určení disparity všechny vypočítané hodnoty v celém procesu.

Rozdělení těchto dvou skupin algoritmů ovšem není zcela striktní. V mnoha případech implementace algoritmů využívajících globální metody stereo korespondence je běžné, že k určování disparit je využito i algoritmů z kategorie lokálních metod. Dále je možné hlavní dvě skupiny metod stereo korespondence rozdělit do podkategorií (viz Obr. 3.1.8).

U lokálních metod je nejrozšířenější využívání metody blokové korespondence (block match). Je to základní způsob, kdy postupně analyzujeme vstupní obrazy po blocích, podle kterých určujeme disparity v závislosti na analýze všech pixelů v jednotlivém bloku. Tento způsob je ostatně také využíván u algoritmů, které implementuji v programu, který je součástí této práce (ssd,sad,...). Co se týká rozdělení globálních metod na jednotlivé kategorie, zde je situace složitější. Existuje celá řada rozličných algoritmů globálního typu. Na Obr. 3.1.8 jsou uvedeny pouze ty nejběžnější.



Obr. 3.1.8: Dělení algoritmů dle velikosti dat nutných pro určení disparity

Jelikož o lokálních metodách se budu zmiňovat v pozdější kapitole, na tomto místě bych teď rád uvedl základní charakteristiky uvedených globálních metod. Tyto metody jsou oproti lokálním metodám složitější na implementaci (v mnoha případech je totiž zapotřebí implementovat značně složité podpůrné mechanizmy) avšak poskytují kvalitativně lepší výsledky.

Propagace zpráv (Belief propagation)

Hlavní technikou používanou pro tuto metodu je určování pravděpodobnosti, kde by se měla disparita nacházet. Využívá se při tom určitého speciálního druhu neorientovaného grafu MRF, ve kterém jsou data uložena v jednotlivých uzlech.

Tyto uzly si poté navzájem posílají speciální zprávy za využití algoritmu BP. Obsahem těchto zpráv je právě pravděpodobnost zda uzel, který zprávu odeslal poslal hodnotu disparity shodnou s informací, kterou poskytuje uzel příjemce. Princip určení výsledné pravděpodobnosti je poté v tom, že všechny uzly v MRF grafu jsou rozděleny na více a méně důvěryhodné, a pokud uzel s

menší důvěryhodností posílá zprávu uzlu s důvěryhodností vyšší výsledná pravděpodobnost bude vyšší než v opačném případě. Kromě tohoto základního principu existují i další vylepšení této metody, například výpočty realizované na 3 MRF grafech, kdy jeden z grafů zodpovídá za výpočet disparity pixelů, druhý se zaměřuje na zpracování ohraničení objektů (line processing) a poslední na určení okluze[8]. Právě tato algoritmy využívající tuto techniku mají velice kvalitní výstupní disparitní mapy.

Dynamické programování

Základním principem přístupu algoritmů využívajících tuto techniku programování je rozdělení hlavního vyhledávacího problému (tedy nalezení disparity) nad daty v 2D prostoru na sérii jednotlivých menších vyhledávacích problémů ovšem pouze nad 1D prostorem. Takovéto zpracování je poté efektivnější a umožňuje značnou optimalizaci.

Grafové řezy (Graph cuts)

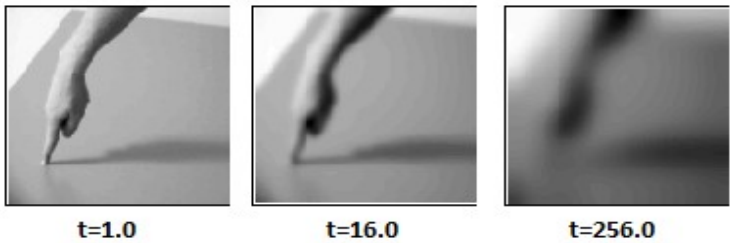
Při použití techniky grafových řezů je analyzován maximální tok v grafu, který je vytvořen ze vstupních dat. Znovu je tedy třeba nejprve vyvinout dostatečně silný algoritmus pro průchod grafem, který nám tento max. tok určí.

Nelineární šíření (Nonlinear diffusion)

Algoritmy využívající nelineární šíření přistupují k řešení problému tak, že akumuluje jednotlivé hodnoty shody, což je odlišné pojetí než tomu bylo u příkladu blokové korespondence u lokálních metod, kdy jsme využívali pouze okno fixní velikosti. Tento druh metody stereo korespondence vyvinuli Daniel Scharstein a Richard Szelinski a podrobnější informace jsou k dispozici v jejich publikaci[9].

Hierarchické metody

Tyto metody, označované také jako scale-space metody využívají pro určení shody přístup ke vstupním obrazům v rozličných stupních. Každý obraz je tak popsán pomocí více jinak vyhlazených obrazů (jiných stupňů), které se liší příslušným parametrem škály (scale parameter) viz Obr. 3.1.9 [10]. Poté co je vstupní obraz takto upraven se již při vyhledávání shody postupuje hierarchicky podle hodnoty parametru škály. Při tomto postupu data získaná z obrazu s vyšším parametrem ovlivňují ty data z obrazu detailnějšího a tím je zaručeno výrazné zredukování prohledávané oblasti v každé úrovni, jelikož nám data z hrubších stupňů upravených obrazů poskytují předběžné informace o pravděpodobné poloze disparity.



Obr. 3.1.9: Totožný obraz při odlišných hodnotách scale-parameter (t)

Tensor voting

Poslední zde uvedenou technikou globálních metod je tensor voting. Tensor můžeme chápat jako podobný objekt jako je vektor či skalár. Tato metoda by se tedy dala volně přeložit jako volení vektoru. Tato metoda byla navržena P. Mordohai a Gerardem Medioni. Tato technika ze vstupních obrazových dat extrahuje některé vlastnosti. Díky tomuto jsou algoritmy využívající tuto metodu poměrně rezistentní vůči chybným výpočtům např. kvůli okolním chybně analyzovaným pixelům.

3.2 Vybrané algoritmy stereo korespondence

V této chvíli bych se rád podrobněji zaměřil na, podle mého názoru, jedny z nejznámějších a nejběžněji používaných algoritmů pro výpočty stereo korespondence. Při výběru těchto algoritmů jsem zaměřil zejména na ty algoritmy, které jsem později zkoušel implementovat a optimalizovat pro výpočty GPGPU a mám s nimi tedy určité vlastní praktické zkušenosti.

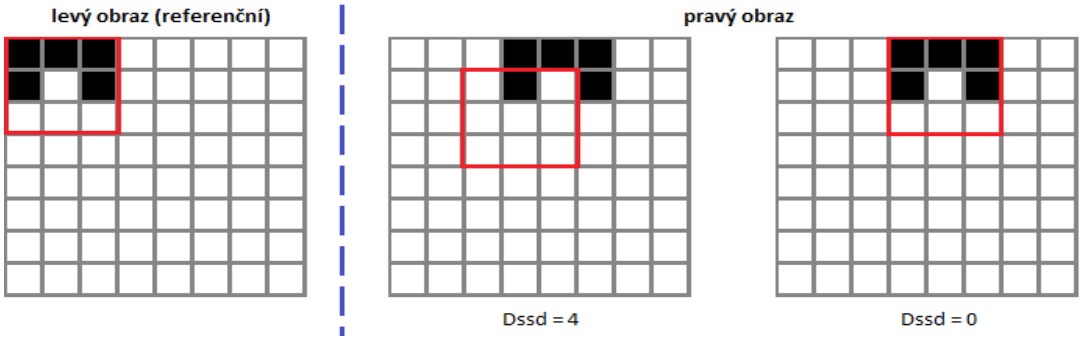
3.2.1 Algoritmus SSD

Tento algoritmus je pravděpodobně jeden z nejznámějších algoritmů stereo korespondence vůbec. Často je pro svou poměrně snadnou implementaci také uváděn pro názorné pochopení určování stereo korespondence.

$$D_{SSD} = \sum_{(i,j)\in U} \left(I_1(x+i, y+j) - I_2(x+i+d_x, y+j+d_y) \right)^2$$

Vzorec 3.2.2: Vzorec algoritmu SSD

Základní princip tohoto algoritmu je v tom, že ohodnotíme určité (předem velikostně stanovené) okolí kolem referenčního pixelu dle jasových složek v referenčním obraze a stejně ohodnocení určitého okolí potom hledáme v obraze druhém. Způsob, jakým lze ohodnotit výše zmíněné okolí referenčního pixelu je matematicky uvedeno viz Vzorec 3.2.2. Nejmenší hodnota D_{SSD}, která nám po analýze celého obrazu vyjde nám označuje pravděpodobnou shodu vybraných okolí v levém a pravém obrazu (viz Obr. 3.2.10) a jsme tedy schopni stanovit velikost disparity pro referenční pixel (viz kapitola 2.2.1).



Obr. 3.2.10: Stanovení hodnoty D_{SSD} pro odlišná okénka ve stejném obraze

Celý vzorec bych v tuto chvíli podrobněji popsal. Klíčové jsou zejména dva parametry i,j. Tyto nám totiž určují velikosti prohledávaného okolí kolem referenčního pixelu se souřadnicemi x,y (dále budeme toto okolí označovat jako okénko). Poté co si na začátku stanovíme velikost okénka, je vhodné ještě vstupní barevné obrazy převést do černobílých, jelikož budeme pracovat pouze s jasovou složkou každého pixelu. Poté se již můžeme pustit do samotného vyhledávání stereo korespondence. Pro úplnost dodám, že v tomto příkladu je jako referenční obraz brán levý vstupní obraz a ve vysvětlení jsem záměrně vynechal jakékoliv možné optimalizace algoritmu.

Vždy budeme procházet pro každý referenční pixel v levém obraze všechny pixely v obrazu pravém(přitom souřadnice každého takového referenčního pixelu jsou ve vzorci označeny proměnnými x,y). Kolem takto stanoveného referenčního pixelu (v levém i pravém obraze) teď musíme analyzovat určitý počet sousedních pixelů v závislosti na velikosti okna.

Analýza jednotlivých pixelů je poté provedena tak, že zjistíme jasovou složku pixelu (na souřadnicích x+i,y+j) v levém obraze a od této hodnoty odečteme jasovou složku pixelu v obraze pravém (na souřadnicích $x+i+d_x,y+j+d_y$). Výsledek poté umocníme číslem 2. Takto projdeme všechny pixely ve stanoveném okénku, a nakonec všechny tyto výsledky sumarizujeme.

Tímto způsobem máme stanovenu hodnotu D_{SSD} pro referenční pixel levého obrazu na souřadnicích x,y k referenčnímu pixelu v obrazu pravém. Postupně posouváme okénko v pravém obraze dále (pomocí zvyšování hodnot d_x,d_y) a stanovujeme další hodnoty D_{SSD}. Až projdeme všechny pixely v pravém obraze, určíme, pro který pixel byla hodnota D_{SSD} nejnižší a zde tedy bude shoda k pixelu (respektive okénka) v levém a pravém obrazu. Celý tento proces dále opakujeme i pro všechny pixely v levém obrazu.

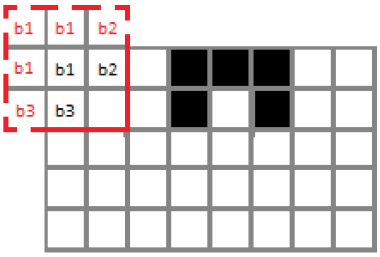
Výše jsem se zmínil o důležitosti správného zvolení velikosti okénka. Právě správná volba tohoto parametru je pro výslednou kvalitu disparitní mapy klíčová (viz Obr. 3.2.11). Na těchto porovnáních je patrné, že při velice malém okénku je ve výsledné disparitní mapě patrné větší množství pixelů u kterých je chybně stanovena disparita, ovšem oproti tomu je ostřeji patrný tvar objektů v obraze.

U větších okének je tomu naopak, chybně určené pixely se eliminují avšak výsledné tvary objektů v obraze již nejsou tak ostré. Při nadměrně velikosti okénka se ovšem znovu v disparitní mapě objevují chybně určené pixely (což je dobře patrné u poslední disparitní mapy, kde se již začínají objevovat bílé stopy, které naznačují, že hodnota disparity nebyla věrohodná).

Obr. 3.2.11: Rozdíly v disparitní mapě při odlišných velikostech okénka

Kromě správně zvolené velikosti okénka se musíme ještě potýkat s dalším problémem, který souvisí právě s okénkem. Při procházením všech pixelů v obrazech se nám totiž u analýzy referenčních pixelů umístěných na kraji obrazu stane to, že část okénka se nám dostane mimo oblast obrazu. Nejsnáze toto lze vyřešit tak, že jasy pixelů, které jsou mimo obraz nastavíme podle jasové složky nejbližšího krajního pixelu (viz Obr 3.2.12).

Jak správně zvolenou velikost okénka tak i problém krajních pixelů samozřejmě není třeba řešit pouze u algoritmu SSD, ale i u ostatních podobných algoritmů stereo korespondence (tedy obecně u lokálních metod).



Obr 3.2.12: Řešení přesahu okna

3.2.1.1 Algoritmus ZSSD

Algoritmus SSD je sice velice oblíbený, avšak má některé poměrně veliké nedostatky. Pokud se nám stane, že se kupříkladu pravý obraz je nasnímaný za malinko jiných podmínek než obraz levý algoritmus SSD bude mít znatelné problémy s určením stereo korespondence. Jelikož porovnáváme pouze jasové složky pixelů v okénku bez jakékoliv další vazby na okolí i třeba opticky méně výrazná změna jasu dovede algoritmu SSD výrazně zhoršit rozpoznávací schopnosti.

V praktickém životě se do situací, kdy je expozice levého a pravého obrazu odlišná se přitom dostáváme velice často. A tak vznikl algoritmus ZSSD, který tuto slabou stránku klasického SSD algoritmu eliminuje. Matematicky je popsán viz Vzorec 3.2.3. Jedinou změnou oproti algoritmu SSD je přidání členů $\overline{I_1(x,y)}, \overline{I_2(x+d_x, y+d_y)}$, které určují průměrnou hodnotu jasové složky v určeném okolí referenčního pixelu. Přidáním těchto členů do výpočtu se poměrně úspěšně dají eliminovat výše zmíněné problémy.

$$D_{ZSSD} = \sum_{(i,j)\in U} [[I_1(x+i, y+j) - \overline{I_1(x,y)}] - [I_2(x+i+d_x, y+j+d_y) - \overline{I_2(x+d_x, y+d_y)}]]^2$$

Vzorec 3.2.3: Vzorec algoritmu ZSSD

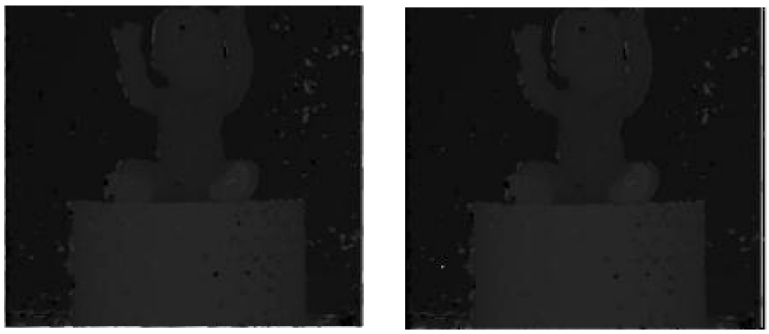
3.2.2 Algoritmus SAD

Druhým nejznámějším algoritmem stereo korespondence je právě algoritmus SAD. Když se podíváme na matematický zápis způsobu vypočítání hodnot DSAD (viz Vzorec 3.2.4) je prakticky totožní s předchozím druhem algoritmu SSD. Jediný rozdíl je v tom, že vypočtený rozdíl jasových složek neumocňujeme dvěma, ale stanovujeme absolutní hodnotu tohoto rozdílu.

$$D_{SAD} = \sum_{(i,j)\in U} | I_1(x+i, y+j) - I_2(x+i+d_x, y+j+d_y) |$$

Vzorec 3.2.4: Vzorec algoritmu SAD

Touto jednoduchou úpravou docílíme nižších výpočetních nároků celého algoritmu, avšak kvalita výsledné disparitní mapy zůstane obdobná jako při využití SSD (viz Obr 3.2.13).



Obr 3.2.13: Rozdílné disparitní mapy při užití SAD a SSD algoritmu

3.2.2.1 Algoritmus ZSAD

Základní algoritmus SAD trpí stejným nedostatkem jako předchozí SSD, tedy tím, že při odlišné hodnotě jasu u levého a pravého vstupního obrazu nedokáže dobře vyhodnocovat stereo korespondenci. Stejně tak i zde existuje podobné řešení tohoto problému jako v předchozím případě. Přidáním klíčových členů do vzorce tento problém eliminujeme (viz Vzorec 3.2.5).

$$D_{ZSAD}=\sum_{(i,\,j)\in U}\left|[I_1(x+i,\,y+j)-\overline{I_1(x,\,y)}]-[I_2(x+i+d_x,\,y+j+d_y)-\overline{I_2(x+d_x,\,y+d_y)}]\right|$$

Vzorec 3.2.5: Vzorec algoritmu ZSAD

3.2.3 Algoritmus Census transform

Zatím jsme se bavili pouze o algoritmech, které pro své výpočty přistupují přímo k jasovým složkám jednotlivých pixelů bez toho, aby je jakkoliv před samotným určením disparity modifikovali. Algoritmus Census transform ovšem patří do skupiny algoritmů, které přímo s jasovou složkou nepracují, i když ji ke své analýze taktéž potřebují.

Tento algoritmus využívá tzv. ne-parametrické lokální transformace, což znamená, že k určování disparity nevyužije přímo samotné hodnoty jasu jednotlivých pixelů v okénku, ale tyto pixely seřadí podle referenční hodnoty určitého pixelu v okénku (obyčejně je to pixel umístěný uprostřed okénka). Ostatní pixely v obrázku se roztřiďují podle toho, jestli je konkrétní zkoumaný pixel (přesněji jeho jasová složka) větší či menší než výše zmíněný pixel referenční.

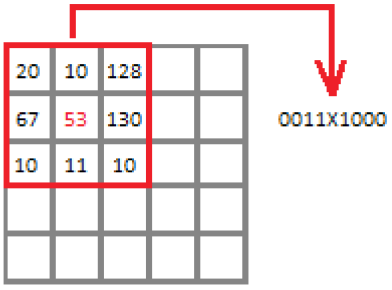
Z každého, takto prozkoumaného okénka ve vstupních obrazech nám vznikne bitový řetězec, který obsahuje 1 na těch místech, kde byla jasová hodnota konkrétního pixelu vyšší nebo rovna referenčnímu centrálnímu pixelu a 0 tam, kde tomu bylo naopak viz Obr.3.2.14 (pixely jsou analyzovány od levého horního rohu k pravému dolnímu a X určuje referenční pixel).

Po tomto kroku ohodnocení okénka pomocí bitového řetězce již jsme schopni porovnávat jednotlivé okénka navzájem a určovat disparitu obdobným způsobem jako tomu bylo u předešlých algoritmů.

Na první pohled se zdá, že se jedná o ideální algoritmus, jelikož řeší největší potíže algoritmů uvedených dříve. Ovšem i tento algoritmus obsahuje jednu, poměrně zásadní slabinu.

Vzniká zde přímá závislost zvolené velikosti okénka s nároky na velikosti alokované paměti pro proměnnou, ve které uložen bitový řetězec. Při velikosti okénka 3x3px nám vzniká bitový řetězec o velikosti 2^3, tedy 8 bitů, zatímco pokud zvětšíme okénko o 2px potřebná velikost alokované paměti pro bitový řetězec se zvýší na 32 bitů.[11]

Zatímco v případech, kdy použijeme algoritmy SSD či SAD jsme volili velikost okénka zejména v závislosti na vlastnostech vstupních obrazů (jejich velikosti,...) v případě využití Census transfom algoritmu musíme při volbě velikosti okénka uvažovat i výše uvedenou vlastnost.



Obr.3.2.14: Princip Census transform

Naproti tomu nabízí tento algoritmus mnoho výhod. Největší z nich je mnohem větší robustnost k méně ideálním vstupním obrazovým datům. Zatímco u dříve uvedených algoritmů nám kupříkladu při změně jasu mezi jednotlivými vstupními obrazy vzrostla chybovost u analýzy disparity,jelikož i ne příliš velkou změnou hodnoty jasu se poměrně výrazně změnila výsledná hodnota ohodnocení okénka, u tohoto algoritmu se změna neprojeví tak zásadně. Navíc v případě použití tohoto algoritmu jsme schopni v obraze rozlišit kupříkladu jevy rotace a odrazu.

Existují samozřejmě i modifikace tohoto základního algoritmu, které jsou optimalizovány pro konkrétní úlohy a poskytují ještě daleko lepší výsledky. Kupříkladu práce [12], kdy autoři modifikovali stávající algoritmus pro detekci tváří.

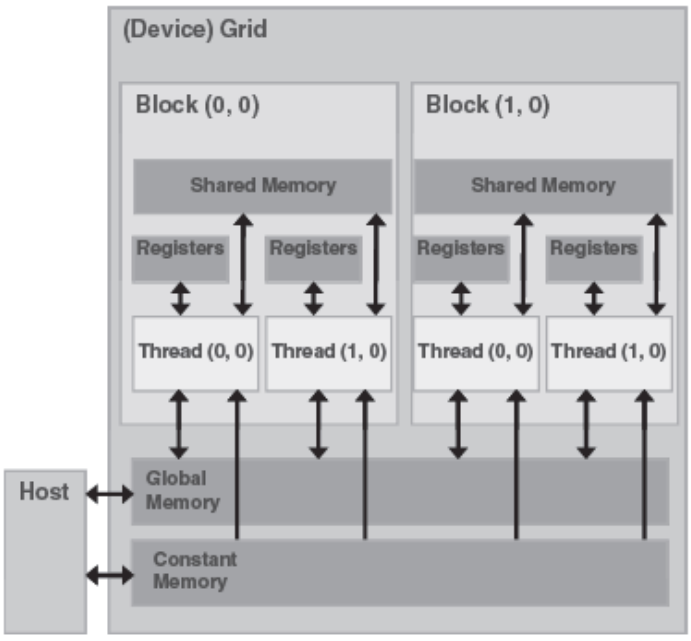
4 Optimalizace algoritmů využitím technologie CUDA

V předchozí kapitole jsme se prozatím zabývali pouze vysvětlením principů nejvyužívanějších algoritmů stereo korespondence. U všech zmíněných algoritmů je na první pohled zřejmé, že výpočetní nároky na analýzu obrazů scény jsou enormní. Když si pro představu vezmeme dnes běžnou velikosti obrázku je nasnadě přemýšlet o optimalizacích algoritmů.

Již v kapitole 2.1 jsem poukazoval na výrazně rychlejší vykonávání výpočetních operací v případě použití grafických procesorů místo klasických CPU. Tento výrazný fakt samozřejmě nezůstal bez odezvy samotných výrobců grafických čipů, kteří pro své GPU začali vyvíjet speciální architektury, které jsou schopny poskytnout vývojářům přístup k vývoji GPGPU aplikací. Mezi dnes nejznámější výrobce grafických karet patří bezesporu i firma NVIDIA, která na počátku roku 2007 představila svou nově vyvinutou architekturu známou pod zkratkou CUDA[13] a právě o této technologii a o optimalizaci algoritmů s pomocí této architektury bych se v následující kapitole zabýval.

4.1 Základní rysy CUDA architektury a požadavky na vývoj

Vývoj aplikací na architektuře CUDA si nelze představit pouze v přenesení výpočetních operací z CPU na GPU. Tato architektura je založena především na paralelním programování a tyto 2 věci, tedy výpočty prováděné přímo na GPU a paralelní přístup k řešení úloh jsou právě tou velice silnou stránkou celého principu vývoje aplikací GPGPU.



Obr. 4.1.15: Model paměti CUDA architektury

Charakteristikou vývoje na CUDA architektuře je dále zejména to, že si musíme vždy uvědomit, že jakékoliv proměnné, které hodláme využívat při výpočtech na GPU (ať již se jedná o pomocné parametry či samotná obrazová data) musíme alokovat v paměti grafického procesoru odlišným způsobem než je tomu při vývoji klasických aplikací využívajících pouze CPU. S tímto přichází ovšem i další problémy, zejména tedy fakt, že k datům alokovaným v prostředí GPU se nelze dostat z hostujícího prostředí jiným způsobem, než kopírováním dat přes speciální funkce, které CUDA poskytuje (např. cudamemcpy).

Architektura paměti je podrobněji ukázána na Obr. 4.1.15. Jsou zde i osvětleny charakteristické prvky paralelního přístupu programování (block,thread) o nichž se budu zmiňovat později v této kapitole ovšem základní princip modelu je dobře patrný i tak. Důležité jsou především 2 základní bloky Host a Device.[3]

Blok Host označuje paměť, která je alokována klasicky na CPU, zatímco blok Device vyznačuje paměť alokovanou přímo pro GPU. Je patrné, že z hostujícího kódu se nelze dostat přímo na sdílená data či do registrů, pouze přes globální paměťový blok nebo blok určený pro konstanty, můžeme tedy data pouze přemisťovat z těchto přístupných paměťových bloků. Z kódu, který je psán přímo pro práci v GPU jsme zase naopak schopni zapisovat či číst z registrů, či paměťových bloků určených pro Device s jedinou vyjímkou a ta je v bloku paměti pro konstanty, kde z Device kódu můžeme pouze číst nikoliv zapisovat.

Konkrétně v programu k této práci přiloženém jsou tyto 2 rozdílné kódy, které jsou uloženy v souboru typu .cu, který je přímo pro práci na CUDA určen (v tomto souboru jsou funkce, které nemají být spouštěny na Device většinou značeny klíčovými slovy „**extern "C"**“ před samotnou deklarací funkce), ještě obohaceny o obslužný kód načítání a tvorby obrázků(obsluha obrázků pomocí OpenCV), který je již uložen v klasickém .cpp souboru určeném pro jazyk C++ a z kterého posléze lze volat host funkce ze souboru .cu, které samozřejmě v sobě mohou volat i funkce napsané pro Device.

Obecně se k označení funkcí podle toho na které straně budou vykonávány slouží tzv. type qualifiers[14], což jsou klíčová slova uvedená před samotnou deklarací funkce. Celkem existují 3 typy funkcí, které jsou uvedeny níže.

Type Qualifiers pro funkce

__device__

Funkce označená tímto kvalifikátorem může být volána pouze z Device kódu. Samotná funkce je poté spouštěna na straně Device.

__global__

Tento typ funkce se vyskytuje právě v přiloženém programu. Může být volána pouze z Host kódu (v našem případě je funkce volána z **extern "C"** funkce) a vykonávána může být pouze na straně Device. Volání této funkce je vykonáváno asynchronně.

__host__

Poslední typ funkce je možno volat pouze z Host kódu a je také na této straně vykonáván. Označení těchto typů funkcí je možno uvést tímto kvalifikátorem, avšak pokud funkce není označena ani jedním z kvalifikátorů automaticky je vykonávána právě jako __host__.

Co se týká ostatních charakteristik vývoje aplikací s pomocí CUDA technologie (zejména tedy paralelního programování) tyto bych dále vysvětlil přímo na příkladech optimalizace již dříve popsaných algoritmů stereo korespondence.

Při vývoji aplikací využívajících CUDA technologii musíme samozřejmě splňovat určité hardwarové i softwarové požadavky. Vývoj i nasazení celé aplikace musí probíhat na PC vybaveném grafickou kartou podporující CUDA technologii a musí mít nainstalovány speciální ovladače. Podle použité grafické karty se také může poměrně zásadně měnit výkon celé aplikace, jelikož odlišné typy GPU na podporovaných grafických kartách mají jiné parametry. Tyto rozdíly se netýkají pouze celkového počtu tzv. cuda cores, které jsou zodpovědné za funkci technologie, ale i odlišnou podporou některých výpočetních funkcí, a tak jsou karty podporující CUDA architekturu označovány ještě navíc tzv. Compute Capapability indexem, který umožňuje snadnější roztřídění jednotlivých modelů podle jejich výkonů pro CUDA (viz Tab 4.1.1).

V případě této bakalářské práce byla aplikace k ní přiložená vyvíjena a testována na grafické kartě Nvidia GeForce 9600M GT (tedy jedná se o kartu pro laptopy), která disponuje celkem 32 cuda cores, s počtem 4 multiprocesorů(MP) a patří dle své Compute Capability do kategorie 1.1. [15] Tedy její omezení je především v nižším počtu celkových vláken, které můžeme v 1 bloku využívat či menší velikosti CUDA array, které využíváme pro uložení obrazových dat.

technické specifikace	Compute Capability				
	1.0	1.1	1.2	1.3	2.0
max. počet vláken na 1blok	512				1024
max. rozměr 1 bloku (v ose x nebo y)	512				1024
max. rozměr 1 bloku (v ose z)	64				
počet registrů (32b) na MP	8000		16000		32000
max. velikost sdílené paměti na MP	16 KB				48 KB
velikost lokální paměti pro 1 vlákno	16 KB				512 KB
max. šířka CUDA array (1D)	8192				32768
max. rozměry CUDA array (2D)	65536 x 32768				65536x65536

Tab 4.1.1: Vybrané technické specifikace grafických karet (dle jejich CC)

4.2 Manipulace s obrazovými daty pro práci na GPU

Prvním krokem, který je nutný k tomu, abychom byly schopni optimalizovat vybrané algoritmy stereo korespondence s využitím architektury CUDA je správně nahrát data vstupních obrazů do takových datových typů, se kterými dokáže CUDA správně pracovat. Tímto datovým typem je cudaArray, což je speciální pole, které nám uchovává všechny potřebné informace o obrazech. Toto pole může být jak 2D tak i 3D, ovšem pro funkci v našem programu budeme využívat pouze klasické 2 dimenzionální pole. Z tohoto datového typu jsme potom schopni vyextrahovat tzv. texture, pomocí které již jsme schopni kupříkladu přistupovat k jasové složce každého pixelu v obraze.

Jak již jsem na počátku této kapitoly zmiňoval, v případě práce s pamětí na straně Device musíme všechny tyto výše uvedené proměnné speciálně inicializovat právě v paměti GPU.

Místo v paměti pro jednoduché datové typy (ať již se bude jednat o jednoduché proměnné či klasické pole) lze alokovat pomocí níže uvedené funkce (Kód 4.2.1).

```
cudaMalloc (void **devPtr, size_t size)
```

Kód 4.2.1: Alokace lineárního bloku paměti v CUDA

Tato funkce alokuje určitý blok lineární paměti o velikosti **size** (v bytech) a ukazatel, který funkci předáváme jako první parametr **devPtr** nastaví tak, aby tuto alokovanou paměť indikoval. Tuto alokaci je vhodné používat pro jakékoliv datové typy, se kterými budeme chtít pracovat v Device kódu, kromě samotného typu cudaArray, který alokujeme za použití mírně odlišné funkce cudaMallocArray (viz Kód 4.2.2).

Nejprve je nutno nastavit formát vstupních obrazových dat, tedy formát v jakém budou jednotlivá data v obraze ukládána do paměti. V tomto případě je vhodné zvolit unsigned char. Poté je již možno zavolat alokaci paměti podle výše zmíněné funkce. Znovu jako v prvním způsobu alokace paměti jako první parametr je uveden ukazatel, který na alokovanou paměť bude odkazovat, další je již zmíněný formát dat a samozřejmě šířka a výška obrazu, pro který tuto paměť alokujeme. Jelikož v tomto případě alokujeme pole, není nutno uvádět velikost paměti v bytech, ale stačí pouze uvést rozměry pole (podle kterého se velikost dopočítá).

```
cudaChannelFormatDesc U8Tex = cudaCreateChannelDesc<unsigned char>();
cudaMallocArray(&cudaImage, &U8Tex, width, height);
```

Kód 4.2.2: Alokace paměti v CUDA pro pole obrazových dat

Tímto způsobem je tedy provedena alokace paměti pro Device kód. Jak již bylo uvedeno v kapitole 4.1, je nutné si uvědomit, že k datům uloženým v takto alokované paměti se lze dostat pouze z funkcí, které jsou určeny k tomu, aby byly vykonávány na straně Device.

Po alokaci je již možné datové struktury naplnit požadovanými vstupními obrazovými daty a získat z nich potřebné informace o každém pixelu. Využít musíme znovu speciální funkce pro kopírování dat, jelikož datovou strukturu máme alokovanou v části Device, ale obrazová data získáváme v klasickém Host kódu. CUDA pro tyto účely disponuje 2 hlavními typy funkcí.

První funkcí je cudaMemcpy, která určena pro kopírování klasických lineárních bloků paměti, tedy v našem programu zejména obsužných proměnných a polí. Pro kopírování polí typu cudaArray jsou ovšem určeny funkce cudaMemcpyToArray a cudaMemcpyFromArray, které jsou chopny efektivně kopírovat celé pole obrazových dat. Tyto 2 základní druhy funkcí mají navíc své speciální druhy, ovšem pro naše účely jsem využíval zejména tyto hlavní.

Pomocí výše uvedených speciálních funkcí, kterými CUDA disponuje jsme tedy schopni efektivně manipulovat se vstupními obrazovými daty přímo s využitím GPU. Níže (Kód 4.2.3) je uveden příklad, jak lze přistupovat k samotným jasovým složkám každého pixelu v obraze.Neprve je nutné nakopírovat obrazová data (zde jsou reprezentována ukazatelem imageData) do cudaArray pole (cudaImage), které je inicializováno v paměti Device.

Zde je jen nutno dodat, že tyto data jsou ve formátu, kdy každý pixel má pouze jasovou složku, tedy jedná se o černobílý obrázek s 1 kanálem. Po úspěšném nakopírování všech dat již jsme shopni pomocí speciálního typu texture přistupovat k jednotlivým pixelům obrazu. Tento typ texture získáme přímo z cudaArray pomocí funkce cudaBindTextureToArray.

K jednotlivým pixelům a jejich jasovým složkám se nakoc lze dostat pomocí funkce tex2D, která nám z texture určitého obrázku vrátí informace o jasové složce pixelu na souřadnicích x,y. Tato funkce je ovšem přístupná pouze z Device kódu, volání odkudkoliv jinud z programu není možné opět z důvodu, že texture, potažmo celé cudaArray je alokováno pouze v paměti Device.

```
texture<unsigned char, 2, cudaReadModeNormalizedFloat> imgTex;

cudaMemcpyToArray(   cudaImage,0,0,imageData,width*height*sizeof(unsignedchar),
                                cudaMemcpyHostToDevice);
imgTex.filterMode = cudaFilterModeLinear;
cudaBindTextureToArray(imgTex, cudaImage);


//nasledujici funkci tex2D lze spoustet pouze z Device kodu
tex2D(imgTex,x,y)
```

Kód 4.2.3: Přístup k obrazovým datům pro určitý pixel

4.3 Zefektivnění algoritmů stereo korespondence

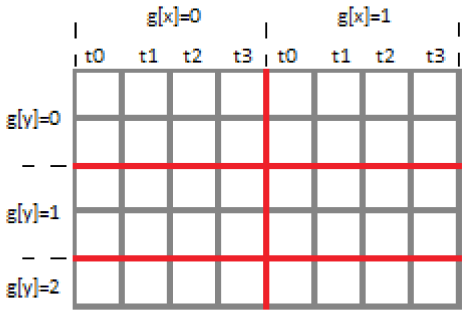
Jednou z velice užitečných vlastností při práci s CUDA architekturou je možnost využití paralelního programování, které nám umožní značně zefektivnit vykonávání náročných výpočetních operací nad obrazovými daty. Navíc ještě efektivnější vykonání můžeme docílit při použití dalších optimalizačních technik (zejména tedy využití tzv. floating window).Díky tomu jsme schopni výsledné určení stereo korespondence provádět v přijatelných časech nejen pro malá vstupní data, ale i pro ty mohutnější.

4.3.1 Rozdělení obrazových dat pro paralelní zpracování

Pro využití paralelního zpracovávání jednotlivých výpočtů nutných k určení disparitních hodnot je potřebné nejprve vstupní obrazová data vhodně rozdělit tak, aby bylo možno vykonávat výpočetní operace co možná nejefektivněji. Technika rozdělení těchto dat je společná pro všechny 3 typy algoritmů stereo korespondence, které jsem ve své práci implementoval.

V prostředí architektury CUDA je takovéto zpracovávání obrazu dobře podporováno. Samotné funkce, které mají být spouštěny na straně Device (tedy kupříkladu __global__ funkce) jsou navrženy způsobem, který předpokládá volání těchto druhů funkcí vícevláknově. Při volání takovýchto funkcí tedy musíme kromě samotných parametrů funkce předávat i speciální datové typy, ve kterých uvádíme, jak množství vláken, které pro zpracování budou využity či velikost sdílených parametrů, tak i tzv. mřížku (grid), díky niž jsme schopni obrazová data rozdělit.

Nám již tedy jen zbývá nalézt způsob, jakým obrazová data rozdělit do jednotlivých sektorů tak, aby je mohli vlákna zpracovávat paralelně a algoritmus probíhal co nejefektivněji. Způsob rozdělení, který je použitý v aplikaci přiložené k této práci je nastíněn na Obr. 4.3.16. Jedná se o princip rozdělení, který je publikován v práci [4], kde aplikován na algoritmus SSD, avšak jeho použití je v hodné i pro algoritmy SAD nebo Census transform.



Obr. 4.3.16: Rozdělení obrazu dle mřížky

Princip tohoto rozdělení spočívá v tom, že si nejprve stanovíme, kolik vláken bude náš algoritmus při vykonávání využívat a tím rozdělíme obraz vertikálně. V oficiálních dokumentech, které Nvidia vydala je uváděno, že by měl tento počet být minimálně 32 (přitom by měl být vždy násobkem 32). Při určení správné hodnoty počtu spolupracujících vláken hraje samozřejmě i roli hardware, na kterém je program spouštěn. V případě grafických karet 8 série se jako optimální uvádí počet 64 až 128, v našem případě jsme program spouštěli na kartě 9 série a nejoptimálnější počet vláken se jevil právě 128. Pro ukázkový příklad na obrázku je ovšem tento počet stanoven na 4.

Z obrázku je patrné to, že každé vlákno zpracovává jednotlivý sloupec pixelů v obrázku, pro který je vytvořeno. Není ovšem vhodné, aby každé vlákno zpracovávalo všechny řádky pixelů ve svém sloupci. Proto musíme ještě k počtu vláken stanovit počet řádků, které bude každé vlákno zpracovávat (tímto rozdělíme obraz horizontálně). Počet těchto řádků je v tomto ukázkovém obrázku roven 2.

Díky tomuto rozdělení jsme poté schopni paralelně vykonávat jednotlivé výpočty s tím, že při volání funkce (typu __global__), která bude zpracovávat obrazová data jen předáme parametry s velikostí mřížky a počtu vláken (viz Kód 4.3.4). CUDA přitom sama spravuje paralelní přístup takže, sama řídí to, aby byla funkce volána pro každé vlákno ve všech blocích mřížky.

```
dim3 grid(1,1,1);        //inicializace promennych grid,threads datoveho typu dim3
dim3 threads(1,1,1);
SHARED_MEM_SIZE    //velikost sdilene pameti na kterou mohou pristupovat vlakna v bloku

threads.x = BLOCK_W; //v nasem pripade pocet vlaken=4
threads.y = 1;           //vlakna jsou pouze 1 rozmerna
grid.x = divUp(width, BLOCK_W);//stanoveni rozmeru mrizky width,height rozmery obrazu
grid.y = divUp(height,ROWSperTHREAD);// funkce divUp deli a zaokrouhluje nahoru

//volani funkce typu __global__
stereoKernel<<<grid,threads,SHARED_MEM_SIZE>>>(--dalsi_potrebne_parametry--)
```

Kód 4.3.4: Rozdělení vstupního obrazu na jednotlivé bloky mřížky

4.3.2 Princip výpočtu disparity v případě paralelního zpracování

Poté, co jsme si obrazová data rozdělili do více bloků takovým způsobem, abychom co nejefektivněji využili paralelní zpracování jsme schopni jednotlivé bloky analyzovat. Princip analýzy každého bloku bude spočívat v tom, že každé vlákno, které v bloku běží analyzuje přidělený sloupec a vypočítává pro něj hodnoty, které jsou později nutné k určení disparity.

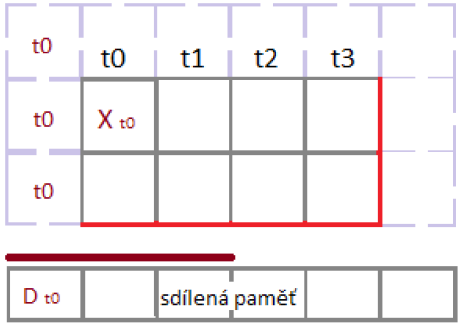
U určování disparity již neplatí, že princip je totožný pro všechny 3 algoritmy, které v programu využívám. Pro algoritmy SSD a SAD je princip stejný, jelikož odlišný je u nich pouze způsob výpočtu konečné hodnoty disparity pro každý pixel, ale v případě algoritmu Census transform jsou odlišná i data, která porovnávám (viz kapitola 3.2.3). Proto je nutné princip výpočtu disparity vysvětlovat pro tyto algoritmy odděleně.

4.3.2.1 Princip výpočtu pro algoritmy SSD a SAD

V případě algoritmu SSD či SAD je princip analýzy jednotlivých pixelů v bloku zobrazen na Obr. 4.3.17. Pro každý blok máme přístupný určený počet vláken, které se starají o provedení výpočtu pro jednotlivé sloupce v bloku. Pro úplnost je nutno dodat, že tento příklad je vytvořen pro okénko velikosti 3x3px.

Nejprve je nutné, aby každé vlákno bylo schopno vypočítat, který pixel je pro něj referenční (v obrázku pro t_0 je to X_{t0}). Toto lze snadno určit, pokud známe souřadnice bloku v mřížce a index konkrétního vlákna. Vše lze ve funkci vykonávané na straně Device (funkce typu__global__) zjistit viz Kód 4.3.5.

Pokud již známe souřadnice referenčního pixelu posuneme tyto souřadnice tak, abychom se dostali na první sloupec okénka. Vlákno posléze zjistí hodnotu jasu pixelu v levém (a poté i pravém) obraze a provede výpočet (dle SSD nebo SAD). Výsledek této operace poté uloží do sdílené paměti na pozici dle svého indexu. Každé vlákno potom stejnou operaci provede i pro zbylé pixely ve sloupci, takže nakonec ve sdílené paměti bude vypočtená hodnota disparity pro celý sloupec.



Obr. 4.3.17: Určení disparity v bloku

Výše zmíněným posunem souřadnic referenčního pixelu ovšem zapříčiníme to, že určitý počet vláken bude muset vykonávat výpočty 2x (konkrétně to budou všechny vlákna, které mají svůj index menší než velikost okénka-1), takže kupříkladu vlákno 0 bude analyzovat sloupec vyznačený na obrázku a navíc sloupec, který obsahuje referenční pixel vlákna 3.

```
x=(blockIdx.x*BLOCK_W + threadIdx.x) //BLOCK_W je pocet vlaken
y=(blockIdx.y*pocetRadku)            //pomoci blockIdx a threadIdx zjistime souradnice
```

Kód 4.3.5: Určení souřadnic referenčního pixelu

Po vykonání tohoto postupu je nutné ještě vlákna synchronizovat, abychom si byly jistí, že všechna data byla dopočítána než se přesuneme k finálnímu určení disparity pro každý pixel. Tento finální výpočet již budeme provádět z dat, které jsou uloženy ve sdílené paměti. Zde opět využijeme vláken a to tím způsobem, že každé vlákno bude procházet n prvků (kde n=velikost okénka) od prvku s indexem, který je stejný jako index vlákna a hodnoty těchto prvků sečte. Tímto způsobem každé vlákno vypočítá hodnotu disparity pro referenční pixel v obraze.

4.3.2.2 Princip výpočtu pro algoritmus Census transform

Základní princip tohoto algoritmu, je naprosto stejný, jako v případě výše uvedených algoritmů. Algoritmus Census transform je rozdílný až v samotné analýze jednotlivých pixelů.

Zatímco v předchozích případech stačilo pouze vyhodnotit samotné jasové složky pixelů z levého a pravého obrazu a jejich rozdíly přímo využívat v tomto případě musíme nejprve každý pixel v bloku levého obrazu porovnávat s určeným referenčním pixelem, poté stejným způsobem porovnávat pixely v pravém obraze a teprve nakonec vyhodnocovat, jestli jsou analyzované pixely shodné či nikoliv (viz Kód 4.3.6).

```
if( leftImage(x,y) < leftImage(xref,yref) left=false;      //funkce leftImage,rightImage
else left=true;                                            //pro nazornost nahrazuji klasicke
if( rightImage(x,y) < rightImage(xref,yref) right=false;   //funkce tex2d
else right true;


if(left-right != 0) sdilenaPamet[thread.idx]+=1;
```

Kód 4.3.6: Princip výpočtu disparity s použitím Census transform

Jak je tedy ze zdrojového kódu patrné, při optimalizovaném algoritmu se již není třeba zaobírat samotným sestavením bitového řetězce pro každé okénko a následné porovnávání těchto bitových řetězců s ostatními, ale přímo pro každý sloupec vypočítává, kolik pixelů v levém obraze má jiný výsledek než v obraze pravém.

4.3.3 Znovuvyužití již jednou vypočtených hodnot

Při použití algoritmů SSD nebo SAD je na první pohled patrné, že jakmile posuneme okénko o 1 pixel dolů většina hodnot, které při novém výpočtu budeme potřebovat již je vypočítána z předchozího kroku a je uložena ve sdílené paměti. Když se podíváme na Obr. 4.3.17, jediné co je potřeba, pokud posunu referenční pixel o 1px směrem dolů je odečíst ze sdílené paměti na určené pozici hodnotu disparity prvního pixelu v prvním sloupci okénka a místo něj přičíst hodnotu 4. pixelu v prvním sloupci okénka.

Výsledkem je tedy to, že namísto analýzy 3 pixelů v obraze analyzuji pouze 1 nový a 1 původní odečítám. Tato úspora se nemusí jevit na první pohled důležitá, ovšem pokud tímto způsobem ušetříme v každém sloupci obrazu dopad na výkon nemusí být zanedbatelný.

5 Porovnání úspěšnosti jednotlivých algoritmů

Pro výsledné porovnání úspěšnosti výše uvedených algoritmů stereo korespondence jsem zvolil testování za pomocí online dostupné aplikace[16] vyvíjené pod záštitou univerzity Middlebury. Hlavním důvodem této volby je především možnost, posoudit úspěšnost vlastních algoritmů s algoritmy jiných vývojářů, kteří v této webové aplikaci sdílí své výsledky s ostatními.

Tato webová aplikace poskytuje testovací obrazová data, pro které je nutné za pomocí vlastních algoritmů vytvořit disparitní mapy a ty posléze analyzovat s referenčními disparitními mapami. O samotnou analýzu disparitních map se poté stará již interní algoritmus aplikace, který funguje na principu porovnávání jednotlivých pixelů obou disparitních map a vyhodnocování, zda jsou pixely označeny za shodné či nikoliv. Na toto vyhodnocování má přitom vliv nastavená povolená odchylka jasu jednotlivých pixelů (v případě všech níže uvedených měření byla nastavena odchylka na 1.0).

Níže uvedené výsledky jsou rozděleny dle využitých algoritmů, přičemž pro každý algoritmus byly pro každou scénu provedeny celkem 4 testy s rozdílnou velikostí okénka pro analýzu každého pixelu.

Důležitým parametrem, který je zde ještě nutno zmínit je velikost testovacích obrazů, jelikož přímo souvisí s výslednými úspěšnostmi všech algoritmů. U scén Cones a Teddy byla velikost obrazu 450x375px, u scény Tsukuba byla velikost 384x288px a u scény Venus 434x384px.

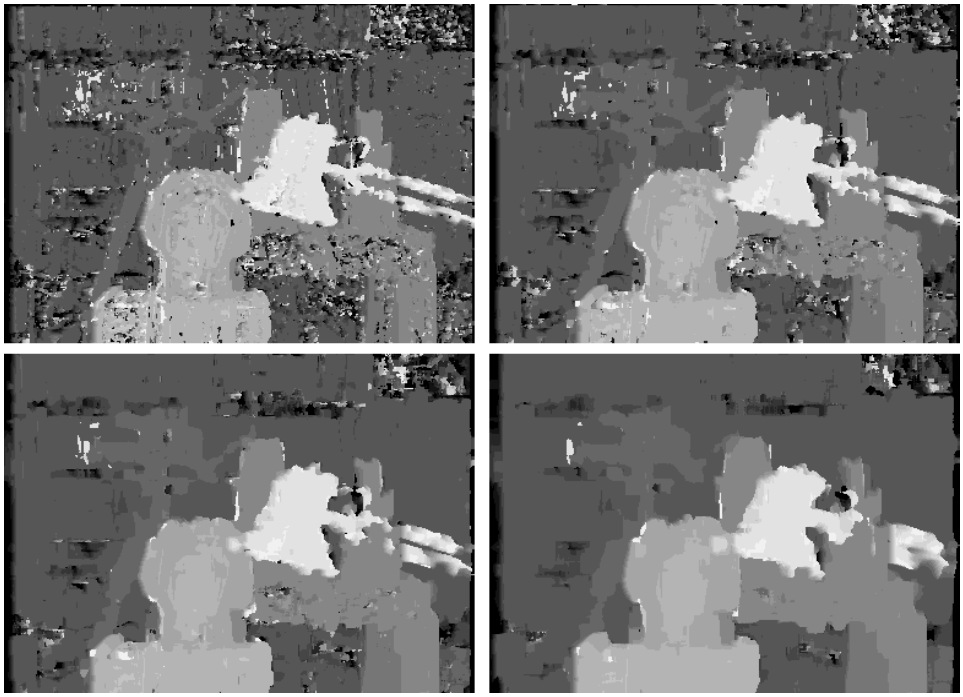
5.1 Úspěšnost algoritmu SSD

V případě využití algoritmu SSD byla jako scéna s nejnižším výskytem chybných pixelů určena Tsukuba scéna. Co se týká vhodné velikosti okénka pro analýzu disparity, nejlepší výsledky algoritmus poskytovat při 7x7px.

Algoritmus SSD se podařilo optimalizovat pomocí CUDA architektury poměrně úspěšně a tak i výsledné časy potřebné k provedení složitých výpočtů jsou velice malé.

okénko[px]	chybné pixely pro jednotlivé scény[%]				p.čas výpočtu[cpu clocks]
	cones	teddy	tsukuba	venus	
3x3	43,0	41,5	19,7	29,4	47
5x5	31,5	34,3	16,7	22,6	47
7x7	27,9	32,1	16,9	20,8	62
11x11	27,6	33,0	19,5	20,6	63

Tab 5.1.2: Úspěšnost tvorby disparitních map s využitím SSD algoritmu



Obr. 5.1.18: Scéna Tsukuba při zvětšujících se velikostech okénka (algoritmu SSD)

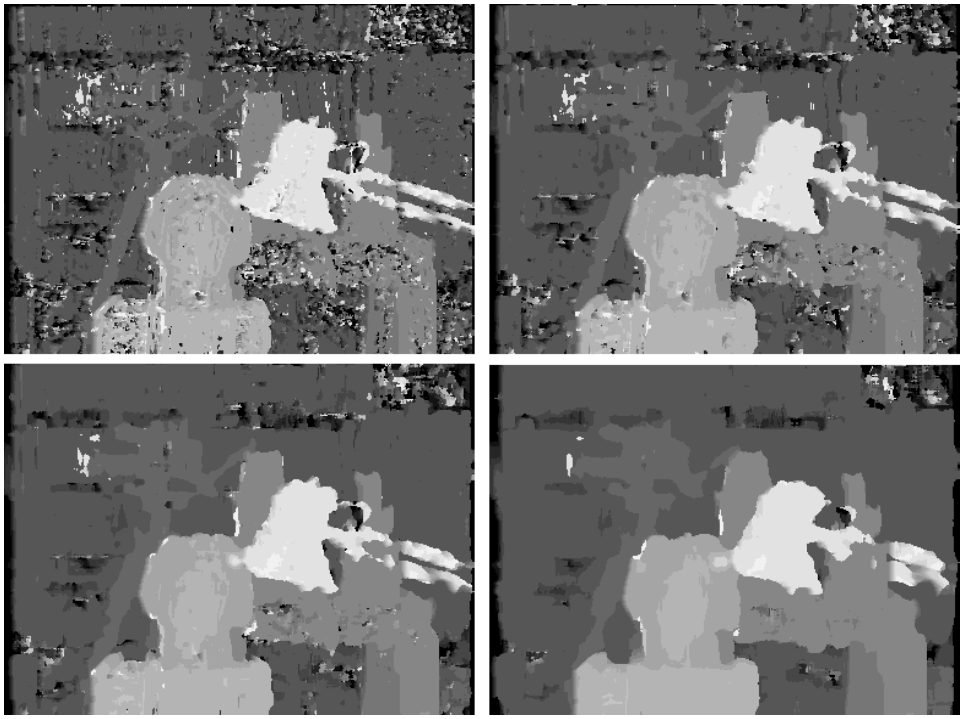
5.2 Úspěšnost algoritmu SAD

Výsledky úspěšnosti při využití algoritmu SAD byly velice podobné jako tomu bylo u předchozího typu algoritmu. Jelikož jediný rozdíl, který mezi algoritmy SSD a SAD je je pouze ve finálním výpočtu ohodnocení konkrétního pixelu (viz kapitola 3.2.2), který je výpočetně méně náročnější v případě algoritmu SAD bylo očekáváno rychlejší provedení celkového výpočtu disparity. Tento očekávaný výsledek se také po provedení testu objevil (viz) a to zejména u analýzy s větší velikostí okénka.

I v případě použití algoritmu SAD se nejkvalitnější disparitní mapa vytvořila ze scény Tsukuba, ovšem rozdíl byl v optimální hodnotě okénka, která v případě tohoto algoritmu byla 11x11px. Se zvyšující se velikostí okénka nad tuto hodnotu ovšem již úspěšnost opět poměrně strmě klesala, a proto se větší hodnoty okénka již v tabulce neobjevují.

okénko[px]	chybné pixely pro jednotlivé scény[%]				p.čas výpočtu[cpu clocks]
	cones	teddy	tsukuba	venus	
3x3	47,0	43,4	19,5	30,1	47
5x5	35,4	35,4	15,4	22,8	62
7x7	30,7	32,6	14,6	20,2	62
11x11	27,5	31,8	15,9	19,4	78

Tab 5.2.3: Úspěšnost tvorby disparitních map s využitím SAD algoritmu



Obr. 5.2.19:Scéna Tsukuba při zvětšujících se velikostech okénka (algoritmu SAD)

5.3 Úspěšnost algoritmu Census transform

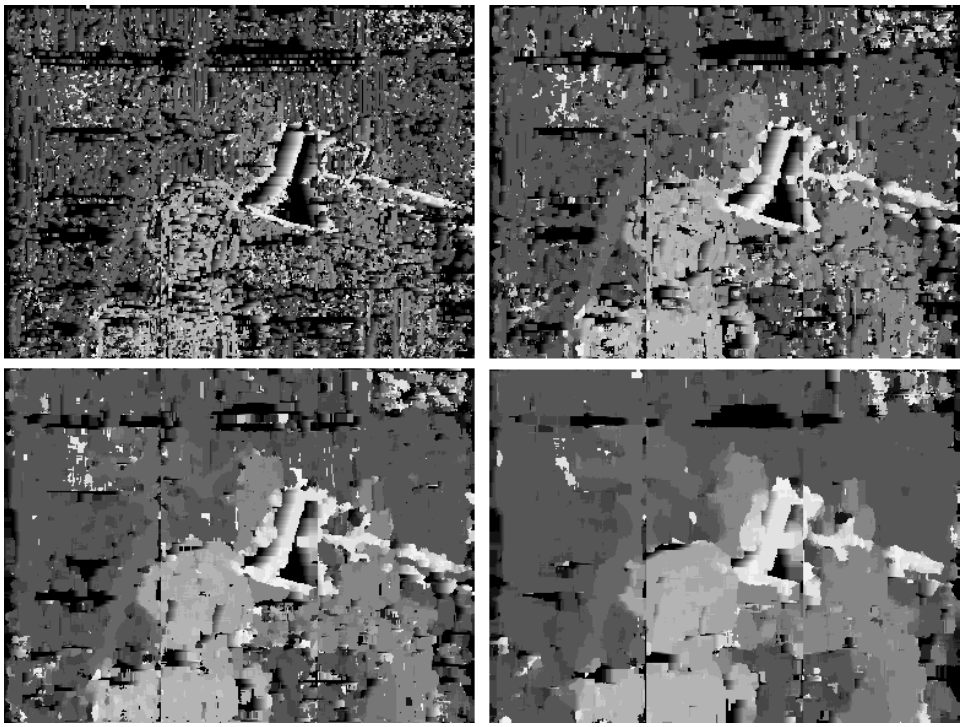
V případě posledního implementovaného algoritmu stereo korespondence byla výsledná úspěšnost již poměrně odlišná, než tomu bylo v předchozích 2 případech. U předchozích algoritmů sice platilo, že s postupně zvětšující se velikostí okénka se snižuje chybovost jednotlivých pixelů (až do kritické velikosti, po které znovu chybovost vzrůstá), ovšem rozdíl v chybovosti nejmenšího a největšího okénka byl nejznatelnější právě při využití algoritmu Census transform. V případě scény Tsukuba dokonce chybovost při určování disparity u jednotlivých pixelů klesla z téměř 75% na 35%.

U tohoto algoritmu se ukázala velikost okénka 3x3px jako naprosto nevyhovující, naopak jako nejoptimálnější je jevila opět velikost okénka 11x11px. Nejkvalitnější disparitní mapa byla vytvořena ze scény venus.

okénko[px]	chybné pixely pro jednotlivé scény[%]				p.čas výpočtu[cpu clocks]
	cones	teddy	tsukuba	venus	
3x3	69,5	74,6	54,6	55,7	93
5x5	38.5	50,2	35,3	36,6	124
7x7	29,3	40,5	28,8	27,2	172
11x11	27,7	35,3	23,8	20,7	265

Tab 5.3.4: Úspěšnost tvorby disparitních map s využitím Census transform algoritmu

V čem ovšem algoritmus Census transfom oproti předchozím poměrně zásadně zaostává je rychlost analýzy stereo korespondence. Jelikož v případě tohoto algoritmu nebylo možno využít některé optimalizační nástroje (např. floating window) vykonávání algoritmů se zpomalilo, což se začne projevovat právě v případě použití větší velikosti okénka.



Obr. 5.3.20: Scéna Tsukuba při zvětšujících se velikostech okénka (algoritmu Census t.)

5.4 Výsledné zhodnocení všech algoritmů

V případě zprůměrování výsledků zhodnocení ze všech scén a určení průměrného počtu chybových pixelů v procentech se jako nejúspěšnější algoritmus stereo korespondence jeví algoritmus SSD a naopak nejméně kvalitní algoritmus Census transform. Tato skutečnost (zejména tedy špatný výsledek Census transform) je ovšem ovlivněna faktem, že algoritmus Census transform dosahoval lepších výsledků až při větších rozměrech okénka, tedy kdybychom měřili úspěšnost v případě jiného vzorku velikostí okének výsledek by to změnilo.

algoritmus	chybné pixely pro jednotlivé scény[%]			
	3x3	5x5	7x7	11x11
SSD	33,4	26,3	24,4	25,2
SAD	35,0	27,3	24,5	23,7
Census t.	63,6	40,2	31,5	26,9

Tab 5.4.5: Výsledné úspěšnosti všech algoritmů stereo korespondence

6 Závěr

Cílem této bakalářské práce bylo zejména seznámení se s principy zpracování obrazových dat, jejich využití při analýze stereo korespondence a následném využití paralelního přístupu k řešení výpočetně náročných úloh s využitím architektury CUDA. Díky této práci jsem nejen získal větší znalosti o problematice tvorby disparitních map, které jsou nedílnou součástí při samotné rekonstrukci 3D scén, ale také jsem se prakticky seznámil s problematikou vývoje aplikací využívajících výpočetních výkonů grafických procesorů. Možnost prakticky se seznámit s takovým způsobem vývoje aplikací pro mne byla velice přínosná, jelikož případy, kdy lze tento způsob optimalizace programů se nevyskytují pouze v aplikacích, které manipulují s obrazovými daty.

Co se týká samotného vývoje algoritmů stereo korespondence optimalizovaných s využitím architektury CUDA, pro něj jsem nakonec zvolil algoritmy SSD,SAD a Census transform. Samotná optimalizace probíhala nejlépe právě u prvních dvou zmíněných algoritmů, což se samozřejmě také v konečném výsledku projevilo na úspěšnosti těchto algoritmů v případě tvorby disparitních map. U algoritmu Census transform bylo největším problémem zvolit správnou strategii rozložení obrazových dat k pozdějšímu paralelnímu přístupu. Nakonec se jako nejlepší řešení ukázalo upustit od samotné tvorby bitových řetězců a jejich následného porovnávání a zvolit přístup obdobný, jako tomu bylo u algoritmů SSD a SAD, tedy rozdělit vstupní obrazy do jednotlivých bloků a pixely pak zpracovávat po sloupcích. Výsledná kvalita disparitních map v případě použití Census transfom algoritmu byla ovšem uspokojivá až při větších velikostech okénka, což mělo ale vliv na celkovou výpočetní náročnost a tak tento algoritmus ve výsledných testech neuspěl tak, jako výše zmíněné algoritmy.

Možnosti dalších optimalizací algoritmů s využitím technologie CUDA jsou ovšem stále veliké. Co se týká algoritmů typu SSD či SAD tak jsou optimalizace provedeny už teď na velice vysoké úrovni, ovšem u algoritmu Census transform je možnost vylepšování, a to především techniky rozdělení vstupních dat do takových oblastí, které poskytnou daleko lepší paralelismus než v případě užití výše zmíněného přístupu. Taktéž je možnost dále optimalizovat algoritmus Census transform tak, aby se snažil využívat již analyzované hodnoty pixelů obdobně, jako to provádějí algoritmy SSD a SAD. Například pokud by se implementovala upravená forma floating window tak, aby algoritmus zkoumal, jestli je nový referenční pixel totožný jako pixel předchozí, vypočtené hodnoty z prvního kroku by bylo možno zachovat. Jaký vliv to ovšem bude mít na celkový výkon celé aplikace, to je nutno ještě důkladně analyzovat.

7 Seznam použité literatury

1: Ian P. Howard, Brian J. Rogers Binocular Vision and Stereopsis. :Oxford University Press,1995..978-0-19-508476-4
2: Pan Hua, Guo Ge Review of Stereo Vision. :Computer Measurement & Control,2004..
3: David B. Kirk, Wen-mei W. Hwu Programming Massively Parallel Processors : A hands-on Approach. Burlington USA:Morgan Kaufmann,2010..978-0-12-381472-2
4: Joe Stam, Stereo Imaging with CUDA, 2008
5: Bogusław Cyganek, J. Paul Siebert An Introduction to 3D Computer Vision Techniques and Algorithms. :John Wiley & Sons,2009..978-0-470-01704-3
6: H. Tang, Z. Zhu, J. Xiao, Stereovision-Based 3D Planar Surface Estimation for Wall-Climbing Robots, 2009
7: Scharstein D, View Synthesis Using Stereo Vision, 1999
8: J. Sun,H-Y Shum, N-N Zheng Stereo Matching Using Belief Propagation . :ECCV,2002..3-540-47967-8_34
9: D. Scharstein, R. Szeliski, Stereo Matching with Nonlinear Diffusion, 1996
10:[online] Tony Lindeberg, Scale-space: A framework for handling image structures at multiple scales, 4.5.2011,
11: R. Zabih, J. Woodfill, Non-parametric Local Transforms for Computing Visual Correspondence, 2004
12: B. Froba, A. Ernst, Face Detection with the Modified Census Transform,
13: J. Sanders, E. Kandrot CUDA by Example: An Introduction to General-Purpose GPU Programming. :NVIDIA Corporation,2010..978-0-13-138768-3
14: NVIDIA, NVIDIA CUDA C Programming Guide, 2010
15:[online] NVIDIA Corporation, Nvidia CUDA, 4.5.2011, http://www.nvidia.com/object/cuda_gpus.html
16:[online] Daniel Scharstein,Richard Szeliski, StereoVision, 4.5.2011, http://vision.middlebury.edu/stereo/